Lambda ifadeleri (C# Başvurusu)

docs.microsoft.com/tr-tr/dotnet/csharp/language-reference/operators/lambda-expressions

Anonim bir işlev oluşturmak için bir *lambda ifadesi* kullanın. Lambda parametre listesini gövdesinden ayırmak için lambda bildirimi işlecini => kullanın. Bir lambda ifadesi aşağıdaki iki formdan herhangi biri olabilir:

- Gövdesi olarak bir ifadeye sahip olan ifade lambda :

  C#

  ```
  (input-parameters) => expression
  ```

- Gövdesi olarak bir ifade bloğuna sahip olan ifade lambda :

  C#

  ```
  (input-parameters) => { <sequence-of-statements> }
  ```

Lambda ifadesi oluşturmak için, lambda işlecinin sol tarafında (varsa) giriş parametrelerini ve diğer tarafta bir ifade ya da deyim bloğunu belirtirsiniz.

Herhangi bir lambda ifadesi, bir temsilci türüne dönüştürülebilir. Lambda ifadesinin dönüştürülebileceği temsilci türü, parametrelerinin ve dönüş değerinin türlerine göre tanımlanır. Lambda ifadesi bir değer döndürmezse, `Action` temsilci türlerinden birine dönüştürülebilir; Aksi takdirde, `Func` temsilci türlerinden birine dönüştürülebilir. Örneğin, iki parametresi olan ve değer döndüren bir lambda ifadesi bir Action<T1,T2> temsilciye dönüştürülemez. Bir parametreye sahip olan ve bir değer döndüren bir lambda ifadesi, bir Func<T,TResult> temsilciye dönüştürülebilir. Aşağıdaki örnekte, `x => x * x` adlı ve kare değeri döndüren bir parametreyi belirten lambda ifadesi, `x` `x` bir temsilci türü değişkenine atanır:

C#

```
Func<int, int> square = x => x * x;
Console.WriteLine(square(5));
// Output:
// 25
```

Aşağıdaki örnekte gösterildiği gibi, ifade lambdaları da ifade ağacı türlerine dönüştürülebilir:

C#

```
System.Linq.Expressions.Expression<Func<int, int>> e = x => x * x;
Console.WriteLine(e);
// Output:
// x => (x * x)
```

Lambda ifadelerini, örneğin, Task.Run(Action) arka planda yürütülmesi gereken kodu geçirmek için yönteme bir bağımsız değişken olarak temsilci türleri veya ifade ağaçları örnekleri gerektiren herhangi bir kodda kullanabilirsiniz. Aşağıdaki örnekte gösterildiği gibi, C# ' de LINQyazdığınızda Lambda ifadelerini

de kullanabilirsiniz:

C#

```
int[] numbers = { 2, 3, 4, 5 };
var squaredNumbers = numbers.Select(x => x * x);
Console.WriteLine(string.Join(" ", squaredNumbers));
// Output:
// 4 9 16 25
```

sınıfındaki yöntemini çağıran yöntem tabanlı söz dizimi (örneğin, LINQ to Objects ve LINQ to XML kullanıldığında, parametre Enumerable.Select bir temsilci t System.Linq.Enumerable type'dır. System.Func<T,TResult> sınıfında yöntemini Queryable.Select çağırarak System.Linq.Queryable , örneğin LINQ to SQL, parametre türü bir ifade ağacı t t `Expression<Func<TSource,TResult>>` değeridir. Her iki durumda da parametre değerini belirtmek için aynı lambda ifadesini kullanabilirsiniz. Bu, `Select` lambdalardan oluşturulan nesnelerin türü farklı olsa da iki çağrıyı benzer şekilde gösterir.

İfade lambdaları

işlecinin sağ tarafındaki ifadeye sahip bir lambda `=>` ifadesi, lambda *ifadesi olarak adlandırılan bir ifadedir.* Bir lambda ifadesi, ifadenin sonucunu verir ve aşağıdaki temel biçimi alır:

C#

```
(input-parameters) => expression
```

Lambda ifadesinin gövdesi bir yöntem çağrısına sahip olabilir. Ancak, .NET Ortak Dil Çalışma Zamanı (CLR) bağlamı dışında değerlendirilen ifade ağaçları oluşturuyorsanız(örneğin, SQL Server'de), lambda ifadelerinde yöntem çağrılarını kullanmamanız gerekir. Yöntemlerin .NET Ortak Dil Çalışma Zamanı (CLR) bağlamı dışında bir anlamı yoktur.

Deyim lambdaları

Deyim lambda, deyimlerinin küme ayraçları içine alınmış olduğu dışında bir ifade lambda'sı ifadesine benzer:

C#

```
(input-parameters) => { <sequence-of-statements> }
```

Bir lambda deyiminin gövdesi herhangi bir sayıda deyimden oluşabilir; ancak, uygulamada genellikle iki veya üçten fazla değildir.

C#

```
Action<string> greet = name =>
{
    string greeting = $"Hello {name}!";
    Console.WriteLine(greeting);
};
greet("World");
// Output:
// Hello World!
```

İfade ağaçları oluşturmak için deyim lambdaları kullanılamaz.

Lambda ifadesinin giriş parametreleri

Bir lambda ifadesinin giriş parametrelerini parantez içine alırsiniz. Boş ayraçlarla sıfır giriş parametrelerini belirtin:

C#

```
Action line = () => Console.WriteLine();
```

Bir lambda ifadesi yalnızca bir giriş parametresine sahipse parantezler isteğe bağlıdır:

C#

```
Func<double, double> cube = x => x * x * x;
```

İki veya daha fazla giriş parametresi virgülle ayrılır:

C#

```
Func<int, int, bool> testForEquality = (x, y) => x == y;
```

Bazen derleyici giriş parametresi türlerini çıkaramaz. Türleri aşağıdaki örnekte gösterildiği gibi açıkça belirtesiniz:

C#

```
Func<int, string, bool> isTooLong = (int x, string s) => s.Length > x;
```

Giriş parametresi türlerinin hepsi açık veya hepsi örtülü olmalıdır; aksi takdirde, cs0748 derleyici hatası oluşur.

C# 9.0'dan itibaren, bir lambda ifadesinin ifadede kullanılmayan iki veya daha fazla giriş parametresini belirtmek için atmaları kullanabilirsiniz:

C#

```
Func<int, int, int> constant = (_, _) => 42;
```

Lambda atma parametreleri, bir olay işleyicisi sağlamakiçin bir lambda ifadesi kullandığınızda yararlı olabilir.

Not

Geriye dönük uyumluluk için, yalnızca tek bir giriş parametresi adlandırılmışsa, `_` bir lambda ifadesi içinde `_` Bu parametrenin adı olarak değerlendirilir.

Zaman uyumsuz Lambdalar

Async ve await anahtar sözcüklerini kullanarak zaman uyumsuz işleme içeren lambda ifadeleri ve deyimlerini kolayca oluşturabilirsiniz. Örneğin, aşağıdaki Windows Forms örnek, zaman uyumsuz bir yöntemi çağıran ve bekleden bir olay işleyicisi içerir `ExampleMethodAsync` .

C#

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        button1.Click += button1_Click;
    }

    private async void button1_Click(object sender, EventArgs e)
    {
        await ExampleMethodAsync();
        textBox1.Text += "\r\nControl returned to Click event handler.\n";
    }

    private async Task ExampleMethodAsync()
    {
        // The following line simulates a task-returning asynchronous process.
        await Task.Delay(1000);
    }
}
```

Zaman uyumsuz lambda kullanarak aynı olay işleyicisini ekleyebilirsiniz. Bu işleyiciyi eklemek için `async` Aşağıdaki örnekte gösterildiği gibi Lambda parametre listesinden önce bir değiştirici ekleyin:

C#

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        button1.Click += async (sender, e) =>
        {
            await ExampleMethodAsync();
            textBox1.Text += "\r\nControl returned to Click event handler.\n";
        };
    }

    private async Task ExampleMethodAsync()
    {
        // The following line simulates a task-returning asynchronous process.
        await Task.Delay(1000);
    }
}
```

Zaman uyumsuz yöntemlerin nasıl oluşturulacağı ve kullanılacağı hakkında daha fazla bilgi için bkz. Async ve await Ile zaman uyumsuz programlama.

Lambda ifadeleri ve tanımlama grupları

C# 7,0 ile başlayarak, C# dili Tanımlama gruplarıiçin yerleşik destek sağlar. Bir lambda ifadesine bağımsız değişken olarak bir tanımlama grubu sağlayabilirsiniz ve lambda ifadeniz de bir tanımlama grubu döndürebilir. Bazı durumlarda, C# derleyicisi demet bileşenleri türlerini belirlemede tür çıkarımı kullanır.

Bir tanımlama grubu, bileşenlerinin virgülle ayrılmış bir listesini parantez içine alarak tanımlarsınız. Aşağıdaki örnek, her bir değeri iki katına çıkarır ve çarpma 'un sonucunu içeren üç bileşeni olan bir tanımlama grubu döndüren bir lambda ifadesine bir dizi sayıyı geçirmek için üç bileşeni olan tanımlama grubunu kullanır.

C#

```
Func<(int, int, int), (int, int, int)> doubleThem = ns => (2 * ns.Item1, 2 * ns.Item2, 2 *
ns.Item3);
var numbers = (2, 3, 4);
var doubledNumbers = doubleThem(numbers);
Console.WriteLine($"The set {numbers} doubled: {doubledNumbers}");
// Output:
// The set (2, 3, 4) doubled: (4, 6, 8)
```

Normalde, bir tanımlama grubu alanları, `Item1` `Item2` , vb. olarak adlandırılır. Ancak, aşağıdaki örnekte olduğu gibi adlandırılmış bileşenlerle bir tanımlama grubu tanımlayabilirsiniz.

C#

```
Func<(int n1, int n2, int n3), (int, int, int)> doubleThem = ns => (2 * ns.n1, 2 * ns.n2, 2 *
ns.n3);
var numbers = (2, 3, 4);
var doubledNumbers = doubleThem(numbers);
Console.WriteLine($"The set {numbers} doubled: {doubledNumbers}");
```

C# tanımlama bilgileri hakkında daha fazla bilgi için bkz. demet türleri.

Standart sorgu işleçleri ile Lambdalar

Diğer uygulamalar arasında LINQ to Objects, türü genel Temsilciler ailesinden olan bir giriş parametresine sahiptir Func<TResult> . Bu temsilciler, giriş parametrelerinin sayısını ve türünü ve temsilcinin dönüş türünü tanımlamak için tür parametreleri kullanır. `Func` temsilciler, bir kaynak veri kümesinde her öğeye uygulanan kullanıcı tanımlı ifadeleri kapsüllemede çok yararlıdır. Örneğin, temsilci türünü Func<T,TResult> düşünün:

C#

```
public delegate TResult Func<in T, out TResult>(T arg)
```

Temsilci, giriş parametresi olan ve `Func<int, bool>` dönüş değeri olan bir örnek olarak örneği olarak örnek olarak örneği `int` `bool` olabilir. Dönüş değeri her zaman son tür parametresinde belirtilir. Örneğin, iki giriş parametresi olan ve ve `Func<int, string, bool>` dönüş `int` `string` türüne sahip bir temsilci `bool` tanımlar. Aşağıdaki temsilci çağrıldığında, giriş parametresinin beşe eşit olup olmadığını belirten `Func` Boole değerini döndürür:

C#

```
Func<int, bool> equalsFive = x => x == 5;
bool result = equalsFive(4);
Console.WriteLine(result);   // False
```

Ayrıca, bağımsız değişken türü bir olduğunda bir lambda ifadesi sebilirsiniz. Örneğin, türünde tanımlanan standart Expression<TDelegate> sorgu işleçlerinde. Queryable Bir bağımsız değişken Expression<TDelegate> belirttiğinizde, lambda bir ifade ağacına derlendi.

Aşağıdaki örnek standart sorgu Count işleci kullanır:

C#

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int oddNumbers = numbers.Count(n => n % 2 == 1);
Console.WriteLine($"There are {oddNumbers} odd numbers in {string.Join(" ", numbers)}");
```

Derleyici giriş parametresinin türünü çıkarabilir veya bunu açıkça belirtebilirsiniz. Bu lambda ifadesi, ikiye bölündüklerinde kalanı 1 olan tamsayıları (`n`) sayar.

Aşağıdaki örnek, dizide 9'dan önce gelen tüm öğeleri içeren bir dizi üretir çünkü bu, dizide koşulu karşılamayan `numbers` ilk sayıdır:

C#

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
var firstNumbersLessThanSix = numbers.TakeWhile(n => n < 6);
Console.WriteLine(string.Join(" ", firstNumbersLessThanSix));
// Output:
// 5 4 1 3
```

Aşağıdaki örnek, bunları parantez içine alan birden çok giriş parametresini belirtir. yöntemi, dizide değeri dizide sıra konumundan küçük olan bir sayıyla `numbers` karşılaşana kadar dizide yer alan tüm öğeleri döndürür:

C#

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
var firstSmallNumbers = numbers.TakeWhile((n, index) => n >= index);
Console.WriteLine(string.Join(" ", firstSmallNumbers));
// Output:
// 5 4
```

Lambda ifadelerinde tür çıkarım

Lambda yazarken, derleyici türü lambda gövdesine, parametre türlerine ve C# dil belirtimlerinde açıklandığı gibi diğer faktörlere göre çıkarayana kadar genellikle giriş parametreleri için bir tür belirtmenize gerek yok. Standart sorgu işleçlerinin çoğunda ilk giriş kaynak dizisindeki öğelerin türüdür. sorgusunu sorgularsanız, giriş değişkeninin bir nesnesi olduğu kabul eder ve bu da yöntemine ve özelliklerine erişiminiz `IEnumerable<Customer>` `Customer` olduğu anlamına gelir:

C#

```
customers.Where(c => c.City == "London");
```

Lambdalar için tür çıkarı için genel kurallar aşağıdaki gibidir:

- Lambda temsilci türüyle aynı sayıda parametre içermelidir.
- Lambdadaki her giriş parametresi, denk gelen temsilci parametresine dolaylı olarak dönüştürülebilir olmalıdır.
- Lambdanın (varsa) dönüş değeri örtük olarak temsilcinin dönüş türüne dönüştürülebilir olmalıdır.

Ortak tür sisteminin hiçbir "lambda ifadesi" kavramı olmadığından, lambda ifadelerinin bir tür olmadığını unutmayın. Ancak, bazen bir lambda ifadesinin "tür" i resmi olarak konuşmak yararlı olabilir. Bu durumlarda tür, Expression lambda ifadesinin dönüştürüldüğü temsilci türüne veya türüne başvurur.

Lambda ifadelerinde dış değişkenlerin ve değişken kapsamının yakalanması

Lambdalar, *dış değişkenlere* başvurabilir. Bunlar, lambda ifadesini tanımlayan yöntemde veya lambda ifadesini içeren türde kapsamda kapsam içinde olan değişkenlerdir. Bu şekilde tutulan değişkenler, aksi halde kapsam dışına çıkacak ve çöp olarak toplanacak olsalar dahi kullanılmak üzere lambda ifadesinde saklanır. Bir lambda ifadesinde tüketilebilmesi için öncelikle mutlaka bir harici değişken tayin edilmelidir. Aşağıdaki örnek bu kuralları gösterir:

C#

```
public static class VariableScopeWithLambdas
{
    public class VariableCaptureGame
    {
        internal Action<int> updateCapturedLocalVariable;
        internal Func<int, bool> isEqualToCapturedLocalVariable;

        public void Run(int input)
        {
            int j = 0;

            updateCapturedLocalVariable = x =>
            {
                j = x;
                bool result = j > input;
                Console.WriteLine($"{j} is greater than {input}: {result}");
            };

            isEqualToCapturedLocalVariable = x => x == j;

            Console.WriteLine($"Local variable before lambda invocation: {j}");
            updateCapturedLocalVariable(10);
            Console.WriteLine($"Local variable after lambda invocation: {j}");
        }
    }

    public static void Main()
    {
        var game = new VariableCaptureGame();

        int gameInput = 5;
        game.Run(gameInput);

        int jTry = 10;
        bool result = game.isEqualToCapturedLocalVariable(jTry);
        Console.WriteLine($"Captured local variable is equal to {jTry}: {result}");

        int anotherJ = 3;
        game.updateCapturedLocalVariable(anotherJ);

        bool equalToAnother = game.isEqualToCapturedLocalVariable(anotherJ);
        Console.WriteLine($"Another lambda observes a new value of captured variable:
{equalToAnother}");
    }
    // Output:
    // Local variable before lambda invocation: 0
    // 10 is greater than 5: True
    // Local variable after lambda invocation: 10
    // Captured local variable is equal to 10: True
    // 3 is greater than 5: False
    // Another lambda observes a new value of captured variable: True
}
```

Lambda ifadelerindeki değişken kapsam için aşağıdaki kurallar geçerlidir:

- Tutulan bir değişkenin kullandığı bellek, ona başvuran temsilcinin kullandığı bellek geri kazanılmaya hazır hale gelinceye kadar geri kazanılmaz.

- Bir lambda ifadesi içinde tanıtılan değişkenler kapsayan yöntemde görünmez.
- Lambda ifadesi kapsayan yöntemden bir ın, refveya Out parametresini doğrudan yakalayamaz.
- Lambda ifadesindeki bir dönüş deyimi kapsayan metodun dönüşmesine neden olmaz.
- Bir lambda ifadesi, bu sıçrama deyiminin hedefi lambda ifade bloğunun dışındaysa bir goto, Breakveya Continue deyimi içeremez. Ayrıca, hedef bloğun içindeyse lambda ifade bloğunun dışında bir sıçrama deyimine sahip olmak için bir hatadır.

C# 9,0 ' den başlayarak `static` lambda ile yerel değişkenlerin veya örnek durumunun istenmeden yakalanmasını engellemek için bir lambda ifadesine değiştiricisini uygulayabilirsiniz:

C#

```
Func<double, double> square = static x => x * x;
```

Statik lambda, kapsayan kapsamların yerel değişkenlerini veya örnek durumunu yakalayabilir, ancak statik üyelere ve sabit tanımlara başvurabilir.